**Report:  
  
Summary of Main Algorithms Used to Tackle the ACW:**

***The boat moves horizontally on the water from side to side in the window (e.g. from left to right, then right to left, and so on).***

Used a while loop (condition – true) that only breaks once an acceptable direction is set. Within the loop if the direction is move left then \_xPos -= velocity\*dt; Is applied. If the direction is move right then \_xPos += velocity\*dt; is applied.

***The cannon’s elevation/pitch is controllable by using the up/down cursor keys.***

For this the angle is increased/decreased but is also checked against the boundaries to make sure a valid elevation/pitch is set.

***A cannonball can be fired from the cannon in the direction that the cannon is pointing by using the space bar. For this minimum specification, the cannon ball can travel in a straight line. The cannonball class must be extended from a base class called NewtonianObject.***

For the first part, the cannon’s angle is calculated and the x and y velocity of the cannonball is then set like so - \_xVelocity = \_initialXVelocity \* (sinf(cannonAngle));

\_yVelocity = \_initialYVelocity \* (cosf(cannonAngle));  
For the second part the cannonball class is extended from the base class NewtonianObject.

***The NewtonianObject class has at least a pure virtual function called Update(float dt). This function must be used to update the Cannonball’s position etc.***The cannonballs position is updated like so - \_xPos = \_initialXPos + \_xVelocity \* \_ballTimer.GetCreationTime();

\_yPos = \_initialYPos - \_yVelocity \* \_ballTimer.GetCreationTime() - 0.5 \* \_gravityVelocity \* \_ballTimer.GetCreationTime() \* \_ballTimer.GetCreationTime();  
GetCreationTime gets the time that has elapsed since the cannonball was created, it ensures processor independent animation is present.

***If the cannonball hits the ship then the ship will sink. At which point another ship will sail into view from one of the sides of the screen.***Check for collision (circlebounding collision), if there was a collision the ship sinks, once the ship is sinking another ship will come into view from the right hand side.

***You should also display a frames per second counter and also a timer which shows the amount of times that has elapsed since the program started.***Upon every frame update the TimeThatHasElapsedSinceProgramStart is retrieved and displayed. To calculate the FPS, FPSFinish = clock(), duration+=(fpsFinish-fpsStart)/CLOCKS\_PER\_SEC  
however this value would vary a lot so it is averaged out, frames++, FPS = frames/duration, fpsStart = clock().

***The boat moves at random speeds as it moves across the water.***int velocity = rand() % (20 - 5) + 5 + 1; // between 5 and 20  
  
**List of all Bonus and Novel Elements Implemented**  
The user is provided with the ability to change the power of shot by using the left/right cursor keys. A visual system (text) showing the user their current power of shot.  
  
Implemented more realistic movement to the cannonball’s motion (gravity has been modelled).  
  
Implemented multiple boats based on a base Ship class containing virtual draw functions (two boats, one black, one red, both visually/aesthetically different).  
  
**Design Approach and Class Design:**  
To try and ensure a robust object orientated game I thought about what classes were needed and any usage/dependencies they had.  
I created the header files first to see what member variables and methods where needed.  
I ensured that all methods that did not modify member variables where declared as const, I consider this a good design approach because anyone implementing the class would know which methods would/wouldn’t affect their objects.  
I ensured that all method parameters are passed in by reference (for speed and efficiency reasons) instead of value, but parameters that would not affect the passed in reference were declared as const. I also consider this a good design approach to object orientated programming because any other programmer using the class would know which parameters are/aren’t safe from change.  
For my class design I also tried to keep high cohesion and low coupling. I feel I achieved this as the only dependencies are between the cannonball and cannon classes. This is because a cannonball needs to know its firing angle and initial x and y positions. The only other dependency is Ship and Box classes, this is because the Ship needs to know its left x and right x boundaries, the left x (in this case) been the cliff’s right edge.  
  
**Class Diagram**  
  
*See the file – Gibraltar\_Class\_Diagaram.vsd.*